

基于树型模型的黑盒测试用例自动生成

孙淑香¹, 侯秀萍¹, 蔡文涛²

(1. 长春工业大学 计算机科学与工程学院, 吉林 长春 130012;

2. 启明信息技术股份有限公司 ERP 部, 吉林 长春 130011)

摘 要:软件测试是保证软件质量的重要手段,尤其是自动化测试可以提高测试效率,降低成本。测试用例的自动获取尤其是黑盒测试用例的自动生成是软件测试的关键和难点。在借助边界值和等价划分等数学原理得到的典型值的基础上,结合自定义的树型模型设计了全面覆盖算法和快速搜索有效用例算法两种面向不同方面的算法。并通过在假设的模型下运用两种算法进行测试数据的生成与测试结果的对比。测试结果表明两种算法各有其自身的优势,有可适用的测试场合、轮次。

关键词:软件测试;黑盒测试;测试用例自动生成;两两组合测试

中图分类号:TP311.52

文献标识码:A

文章编号:1673-629X(2009)02-0077-04

Black - Box Testing Case Automatic Generation Based on Tree Model

SUN Shu-xiang¹, HOU Xiu-ping¹, CAI Wen-tao²

(1. Department of Computer Science and Engineering, Changchun University of Technology, Changchun 130012, China;

2. ERP Department of Qiming Information Technology Ltd. Co., Changchun 130011, China)

Abstract: Software test is an important artifice to assure the quality of software, especially test case automatic generation could improve test efficiency and reduce costs. It is sticking and difficult points to automatically generate test cases, especially black - testing cases automatic generation. Presents two methods for black - testing case automatic generation, which are based on the tree model designed, and on the typical data which are gained through the mathematics theories, such as boundary data selection, equivalent compartmentalizing and so on. It also compares the test sequent which is got from the two methods based on the given model. Test results indicate each of the two methods has its own advantage, and has its own test scene and test turns.

Key words: software testing; black - box testing; test case automatic generation; pair combination testing

0 引 言

在软件测试过程中,测试用例的生成是软件测试的关键和难点。多年来很多研究人员对此进行了大量的研究^[1,2]。然而目前,测试用例的生成主要靠手工完成,而且要求软件测试人员具有一定的经验和较高的专业水平。因而,测试过程往往带有很大盲目性,致使测试效率低下,软件成本居高不下,软件质量也很难保证。为此,测试用例的自动生成方法及测试工具的研究有其现实的必要性。

基本的黑盒测试方法包括:等价划分法、边界值分

析法、因果图分析法、错误猜测法等,即使使用这些方法来手动生成覆盖全面的测试用例也是很现实的。

然而穷举测试又是无法实现的,这有两方面的含义:

一是我们无法测试一个程序以确保它是无错的;

二是软件测试中需要考虑的一个基本问题是软件测试的经济学^[3]。

借助边值分析法和等价类划分法等基本黑盒测试方法选取一组典型的值^[4],如何结合上述典型值实现测试用例的自动生成,来达到对所测试系统的全面覆盖是黑盒测试用例自动生成的一个研究方向,另外如何结合典型值尽快找到测试系统的有效用例也是黑盒测试用例自动生成的一个研究方向。这里的有效用例不是能使程序通过的用例,而是能找出程序中存在的错误的用例,因为测试的目的是找到程序中存在的问题而不是证明程序是正确的。

收稿日期:2008-06-05

基金项目:吉林省科技发展计划重大科技攻关项目(吉科合字:20040305)

作者简介:孙淑香(1981-),女,山东烟台人,硕士研究生,研究方向为软件工程;侯秀萍,教授,硕士研究生导师,研究方向为软件工程。

1 基于树型模型的黑盒测试用例自动生成算法

1.1 基本概念及性质

1.1.1 概念

设系统对外接口由 m 个参数 c_1, c_2, \dots, c_m 组成, 它们的取值集合分别记为 T_1, T_2, \dots, T_m , 其中 c_j 参数可取值的个数为 t_j , 而 T_j 是有穷的符号集, $t_j = |T_j|$ 是点集 T_j 中元素的个数。

定义 1 基于接口的测试用例: 设 m 个接口参数 c_1, c_2, \dots, c_m 的某种取值组合为 $u_s = \{u_{s1}, u_{s2}, \dots, u_{sm}\}$, 则 u_s 称为一个测试用例, 所有测试用例的集合记为 A , 则集合中测试用例的个数为 $|A| = t_1 * t_2 * \dots * t_m$ 。

定义 2 测试用例基: 设 $n = \max_{1 \leq j \leq m} \{t_j\}$, 在 A 中选取 n 个测试用例, 构成测试用例组 U_n , 使在 U_n 中每个参数的所有取值都至少出现一次, 则称 U_n 为 A 的测试用例基^[5]。

1.1.2 性质

设 U_n, V_n 是 A 的任两个测试用例基, 则 U_n, V_n 是相互等价的^[5]。

基于上述理论可知, 在待测参数 m 给定的情况下, 每个参数的离散取值集合可以确定(用等价划分法和边界值法可以确定该参数的具有代表性的取值集合), 从而测试用例基中测试用例的个数 n 可以确定。如何获得这 n 个测试用例, 可以在测试用例表的基础上, 对任意两个接口参数两两组合全面覆盖的原则来选择测试用例。

1.2 树型模型的构成及描述

设某个系统其接口参数有 n 个, 第 i 个参数有 t_i 个取值 $i = 1, 2, \dots, n$, 对参数按取值个数的多少顺序排列(即 $t_1 > t_2 > \dots > t_n$)。将这 n 个参数及其取值按如下方式构成倒立的树(见图 1):

1) 树型结构的根节点为算法的入口;

2) 树型结构的第 1 层按如下方法构成: 树型结构的第 1 层节点为 n 个参数中取值个数最多的节点即有 t_1 个取值的参数 C_{i1} , 将 t_1 个取值予以编号, 并将标好号的 t_1 个取值从左到右依次排列在 Entrance 下;

3) 第二层及其以下各层按如下方法构成: 第 2 层以下各层分别为有 t_2, \dots, t_n 个取值的参数 C_{i2}, \dots, C_{in} ; 将参数 C_{ij} 的 t_j 个取值予以编号, 即 $1, 2, \dots, t_j$ 。将编好号的 t_j 个取值作为 $C_{ij}(j > i)$ 的每个取值的叶节点予以连接。

4) 纵观整棵树, 从倒立的树的第一层叶节点开始, 依次往下, 每一层代表一个参数的取值, 并且叶节

点的个数呈下降顺序。

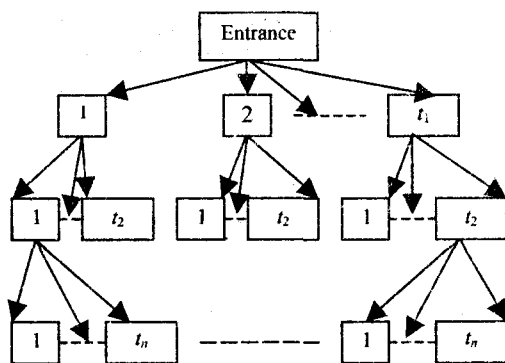


图 1 $n - t_i$ 树型模型

1.3 全面覆盖算法描述

在上述的树型模型下假定 $n = 3, t_1 = t_2 = t_3 = 3$ 。树的第 1 层表示参数 C_{i1} 的 $t_1(t_1 = 3)$ 个取值, 编号为 1、2、3; 第 2 层表示参数 C_{i2} 的 $t_2(t_2 = 3)$ 个取值, 并将这 3 个取值作为 t_1 的叶节点予以连接; 同理编号并连接 2、3 层。从第 1 层某个节点出发沿着树枝的方向向下推进, 直到到达整棵树的叶节点, 形成一条路径, 路径上的节点号序列就代表一个测试用例。1-1-3, 就表示第 1 个参数取第 1 个值, 第 2 个参数取第 1 个值, 第 3 个参数取第 3 个值构成的一个测试用例。

具体算法步骤如下:

1) 计算顺序: 按倒立树的层次从上到下, 每一层从左到右的顺序逐个节点执行。

2) 第 1 层的节点按编号从小到大的顺序依次沿着树杈按其叶节点号递增的次序给每个叶节点发送自己的节点号。

3) 第 2 层每个节点每接收到一个节点号, 就把自己装配在这个节点号的后面, 形成一个节点号串, 并把它按照接收的顺序保存。

4) 第 1 层每个节点发送完毕, 从第 2 层的每个根节点开始按照下面的算法向其每个叶节点发送该节点内形成的号码串: 第 i 层($i \geq 2$) 的第 j 号节点($1 \leq j \leq t_i$) 把该节点内第 1 个组装好的号码串, 与第 $i+1$ 层节点(按编号从小到大顺序) 与每个节点内已经收到的来自其它第 i 层节点发来的每个符号串进行组合;

5) 按照第 i 层中编号从小到大的顺序重复步骤 4 发送过程直至该层节点均将其号码串发送到叶节点, 第 i 层中的每个节点的叶节点均把自己装配到其根节点发送过来的符号串后。

6) 最后一层叶节点为输出层, 每个节点内形成一组节点号串, 每个节点号串就是产生的测试用例, 这些串的全体构成了基于这 n 个参数两两组合测试用例集。

算法:

输入 T_i : C_i 的取值集合;

$n = |T_i|$: C_i 的取值个数;

$\text{String}[] s_1 = \text{new String}[]\{T_1, T_2, \dots, T_n\}$

m : 接口参数的个数;

输出 编号串集合

begin

doIt(int m, int n)

{

if($m == 0$)

{

$\text{String}[] s = \text{new String}[]\{ "", "", "", "" \};$

for($x = 0$; $x < n$; $x++$)

{

$s[x] = s1[0][x];$

}

return s;

}

List list = new ArrayList();

for(int j = 0; j < n; j++)

{

ss = doIt($m - 1$, n);

for($y = 0$; $y < ss.length$; $y++$)

{

list.add($s1[m][j] + "," + (\text{String})ss[y]$);

}

}

return list.toArray(); //输出

}

end

1.4 快速搜索有效用例算法描述

将 n 个接口参数的各个取值予以编号,后按照如下步骤快速地搜索有效的组合,即有效的测试用例。

算法步骤:

1) 从倒立的树的根节点开始,层次按照从上到下,叶节点按照从左到右的方向搜索组合。

2) 第1层编号最小的节点把自己的编号传递给其叶节点中编号最小的节点;

3) 第2层编号最小的节点把自己的编号及其上层编号一起组成的串传递给它的叶节点中编号最小的节点。依次各层节点均取节点编号最小的节点组成数字串,构成一个测试用例。

4) 当按照最小节点原则找到的测试用例不能找到系统存在的缺陷时,依据下面原则进行寻找:

(1) 保留3)中除了第一个编号以外的其他编号组成的数字串,将处于第一位的编号用同根节点下的次小的编号代替,直到遍历完此根节点。

(2) 在遍历完倒数第二层最小编号下的叶节点后,遍历倒数第二层的次小编号(保持与最小编号的根节

点一致)下的叶节点,遍历顺序仍依编号从小到大。

(3) 按照倒推树的原则不断变换不同位置的根节点直到遍历到第一层的编号最大的根节点。

5) 在上述过程中若出现找到系统缺陷的数字串即有效的测试用例则终止搜索过程。

1.5 算法比较

在假定 $n = 3, t_1 = t_2 = t_3$ 时,并且组合 2-3-1 测试用例能测试出系统存在缺陷,运用如上两种算法进行比较(见图2)。

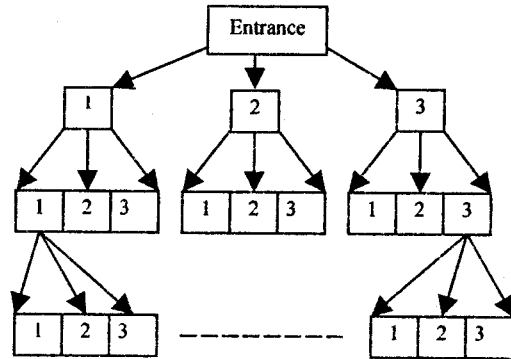


图2 3-3树型结构

运用全面覆盖算法时,需要先将第一层和第二层的各种取值进行组合完全,依次生成数字串:1-1、1-2、1-3、2-1、2-2、2-3、3-1、3-2、3-3;然后将第二层与第三层组合生成数字串:1-1-1、1-1-2、1-1-3、2-1-1、2-1-2、2-1-3、3-1-1、3-1-2、3-1-3、1-2-1、1-2-2、1-2-3、2-2-1、2-2-2、2-2-3、3-2-1、3-2-2、3-2-3、1-3-1、1-3-2、1-3-3、2-3-1、2-3-2、2-3-3、3-3-1、3-3-2、3-3-3。在以上数字组合全部生成后,即可用以上组合所代表的测试用例对待测系统进行测试,当输入2-3-1组合代表的测试用例时显示系统存在缺陷,为了实现在此算法生成的测试用例的全面性,后续的几个组合即测试用例也要对系统进行测试,从而说明系统在除了组合2-3-1代表的测试用例证明系统存在缺陷外,系统此时不存在其他方面的错误。但并不能保证在修改了上述缺陷后上述结论的正确性。回归测试时除了要试验2-3-1所代表的测试用例,同时也要测试其他的组合。

运用快速搜索有效用例算法时依次生成的数字串为:1-1-1、2-1-1、3-1-1;1-2-1、2-2-1、3-2-1;1-3-1、2-3-1(3-3-1);1-1-2、2-1-2、3-1-2;1-2-2、2-2-2、3-2-2;1-3-2、2-3-2、3-3-2;1-1-3、2-1-3、3-1-3;1-2-3、2-2-3、3-2-3;1-3-3、2-3-3、3-3-3);当找到2-3-1时算法终止运行,整棵树不需要继续搜索,即括号里面的数字组合不需要生成,也不需要用来执

行测试系统,从而节省了测试时间,一定程度上提高了测试效率。

当然在运用等价划分法和边界值法得到的基础数据,不论怎么组合都不能发现程序中存在的缺陷时,以上两种算法效果是相同的,都必须生成 t^n 个测试用例。并运用这 t^n 个测试用例对系统进行测试。

全面覆盖算法在接口参数 C_i 的取值个数 t_i 很大时,实现起来比较困难,所以此算法有其适用范围。当在用边值分析法、等价划分法等确定参数 C_i 的代表性取值时,应该尽量减少取值个数,依据测试轮次或测试阶段选取典型值。此算法适于在测试系统的单元即最基础模块,适于在对系统进行压力测试、容错性测试等测试中运用。

快速搜索有效用例算法适于在对系统单元进行可用性测试时运用。可用性测试是在第一轮测试之初,初步测试应用程序,以确定该程序已实现基本需求,即可用。如果基于此来运用该算法时,可尽量减少运用边值分析和等价划分等基本黑盒测试方法所得到的基础参数的个数。

2 结束语

研究了功能测试即黑盒测试的单元测试阶段,针对不同测试目的有效的测试用例自动生成算法。从以上试验结果可知,以上两种基于树型模型的算法各有其优缺点。全面覆盖算法很好地做到了对整棵树的较

全面的覆盖,而快速搜索有效用例算法以树型结构的深度搜索为基础能以较快的速度找到有效的数字组合即测试用例,这里的有效的测试用例不是使测试能通过用例,而是能找到系统中存在缺陷的测试用例。在不同的测试阶段及不同的测试目的时,选择有效的测试方法能大大降低测试成本,提高测试效率。对于有 n 个参数,每个参数有 t 个取值的系统,最好的测试用例设计是对任意两个参数的各种组合只覆盖一次的测试用例集,即在测试用例集中整棵树的节点均只出现一次,且每个树枝只遍历一次。

参考文献:

- [1] DeMillo R. Experimental results from an automatic test case generator[J]. ACM Transactions on Software Engineering Methodology, 1993, 2(2): 109 - 175.
- [2] DeMillo R. Constraint based automatic test data generation [J]. IEEE Transactions on Software Engineering, 1991, 17 (9): 900 - 910.
- [3] Myers G J, Badgett T, Thomas T M, et al. The Art of Software Testing[M]. 王 峰, 陈 杰译. 北京: 机械工业出版社, 2006.
- [4] Chen T Y. A new heuristic for test suite reduction[J]. Information and Software Technology, 1998, 40 (5 - 6): 347 - 354.
- [5] 杨劲涛, 荷 清. 黑盒测试用例基的研究[J]. 计算机工程与科学, 2006, 28(5): 130 - 131.

(上接第 76 页)

将旧数据和新数据区分开。

(4) SDO 可以作为 XML 构件或 Java 对象存在。SDO 可以很方便地对 XML 和 Java 进行转化,并且支持 XSD 中的数据结构。

随着 SOA 应用的逐步推广和深入, SCA/SDO 将会受到越来越多的关注,这将推动其发展和完善。两者相辅相成, SCA/SDO 也势必会推动 SOA 的实施。相信今后必定会有更多的厂商支持 SCA/SDO,其也将成为实施 SOA 的首选。

4 结束语

介绍了 SOA 面向服务架构的理论知识,并运用 SOMA 方法论对服务这一抽象的概念进行分析。最后介绍了 2 个实现 SOA 的重要技术: SCA 与 SDO,分别代表了调用方式的抽象和传输数据的抽象。

总之, SOA 就是一种企业信息管理应用的框架,它着眼于日常的业务应用,并将它们划分为单独的业

务功能和流程,即所谓的“服务”^[3]。形象地说,这些“服务”就像大小不一的乐高玩具中的积木,它们之间的灵活组合可适应不同的业务管理需要。这将成为企业业务信息化的新模式,带给人们更快捷便利的社会服务响应。

参考文献:

- [1] 喻 坚, 韩燕波. 面向服务的计算原理及应用[M]. 北京: 清华大学出版社, 2006.
- [2] Newcomer E. Understanding SOA with Web Services[M]. 徐涵, 译. 北京: 电子工业出版社, 2006.
- [3] 庞引明. 认清 SOA 的本来面目[N]. 计算机世界报, 2005 - 06 - 20(6, 7).
- [4] 王金玲, 朱诗生, 符群卫. 基于 Web 服务的 SOA 软件部署的研究[J]. 现代电子技术, 2007(4): 52 - 55.
- [5] Fielding R T. Architectural Styles and the Design of Network - based Software Architectures[D]. California, USA: California University, 2000.